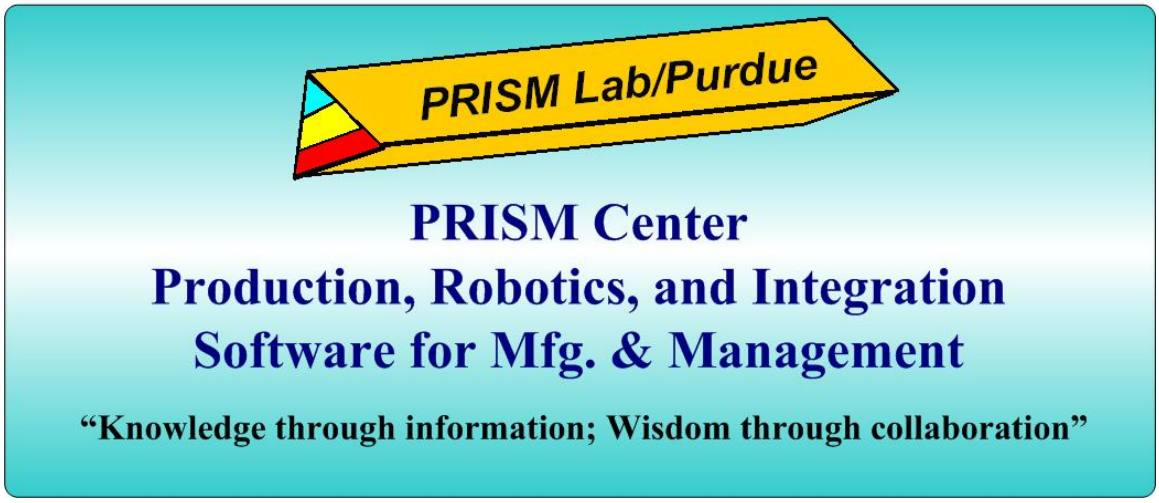
DESIGN OF A TASK PLANNING CONFLICT AND ERROR DETECTION SYSTEM WITH ACTIVE PROTOCOLS AND AGENTS

Chao-Lung Yang, Shimon Y. Nof

PRISM Research Memorandum No. 2004-P1
August, 2004



This research is supported in part by GM R&D Center.

Juan D. Velasquez
Research Series Coordinator
PRISM Center
School of Industrial Engineering
Grissom Hall
315 N. Grant Street
Purdue University
West Lafayette, IN 47907-2023
jvelasqu@exchangepurdue.edu

Design of a Task Planning Conflict and Error Detection System with Active Protocols and Agents

Chao-Lung Yang, Shimon Y. Nof
August 2004

Executive Summary

In this research, an agent-based Conflict/Error Detection Protocol (CEDP) is developed to continuously detect conflict and error problems in a distributed collaborative task-planning environment. By applying Active Middleware architecture, the detection protocol can help detection agents and support their evaluation of the detection process, and exchange detection information among collaborative participants. A measure called “conflict-severity” is developed to evaluate propagated conflict situations and assess the seriousness of the conflict; and if justified, a possible detection. Experiments with CEDP have been run on three basic Co-net forms, linear, tree, and parallel. The experimental results show that the CEDP is able to detect conflict and error propagation in different task dependency networks. A well-design CEDP is important, because an effective communication mechanism to improve the quality of CE detection in a collaborative, distributed network will determine the benefit to the company. Therefore, based on this research, recommended design guidelines for CEDP are developed.

Acknowledgement

This research has been developed with the PRISM Center support, and support from GM R&D Center Project on "Design of Active Middleware for Error and Conflict Detection (2004)". This paper is based in part on C-L. Yang’s M.S.I.E. thesis which was completed in August 2004.

1. Introduction

Distributed collaboration among organizations is common in business and manufacturing nowadays, because it enhances the effectiveness of operations. Combining the expertise and attributes of different participants, and synthesizing their achievements provide enterprises with more flexibility and the ability to overcome challenges in the complex business world. For example, in a supply network, enterprises distribute the tasks by selecting specialized skills from different enterprises. In a distributed design project, for instance, team members contribute their expertise to achieve the common design objective cooperatively. No matter how the collaboration is formed or organized, the goal of each participant involved in the collaboration is to finish the assigned tasks as accurately and efficiently as possible.

However, conflicts and errors (CE) are unavoidable in a distributed collaborative environment. Since every participant in the collaboration might have their own expertise, local resource management, and project-based schedule, conflicts between different participants' expertise, schedule, or operation seem inevitable. In other words, conflict management is necessary in collaboration networks and teamwork projects. Many researchers have provided successful solutions for conflict resolution in a problem-solving environment. Examples include design objective problem-solving for concurrent engineering [Tan et al. 1996, Khanna et al. 1998, Li et al. 2002, Lara and Nof 2003], and exception handling in workflow systems [Hagen and Alonso 2000, Klein 2000, Klein and Dellarocas 2000, Luo et al. 2000, 2003].

Most of the previous work has focused on how to solve CE problems. The findings suggest that the information exchange among Co-Us[1] is crucial. However, few researchers define the type of information that should be shared or exchanged to facilitate conflict detection and how Co-Us communicate with each other from a detection perspective. Finally, once a conflict occurs, a means to measure a conflict has not yet been addressed.

Once a CE happens in a collaborative environment, this unexpected event might also affect other participating members. In our research, we define this situation as *CE propagation*. For example, if a schedule conflict occurs in a member's operation, it might change its operations after rearranging the schedule and solving the local conflict. Yet this updated schedule might also cause other collaborating members to experience schedule conflicts. If we can recognize the "infection area", that is a set of members affected by a detected CE, a better CE resolution/recovery method can be applied to moderate over-all CE problems. Thus, this kind of CE propagation must be detected once an error or conflict occurs and the "infection area" of the CE needs to be located.

To investigate CE problems in a collaborative environment and develop applicable information mechanisms and measures, three related research problems are defined:

a. *Once a conflict or error occurs in a certain autonomous entity, how can this event affect other entities within the coordination network?*

   Once a CE event occurs, if it is not detected and resolved, it might also affect other participants due to the error, or the task dependency between participants. When a CE occurs, it is important to know how to inform or report about it to

[1] A Co-U is a collaborating unit, any member of the Co-net, which is the collaborative network model of the environment as defined below.

responsible members or supervisors. Furthermore, how to locate the conflict/error "infection" area is also important for the resolution/recovery stage. Only after identifying who the problem initiators are and those that will suffer from it, can the negotiation process among CE insiders for resolving the problem be performed effectively.

b. *How can a CE detection protocol (CEDP) be developed and specified?*

   In a distributed environment, an individual entity makes independent decisions based on its limited information. For example, once a CE problem occurs, the announcement information about the CE problem should be circulated among Co-Us in order to caution dependent participants and enhance the overall performance. A Co-U can request other Co-Us to provide CE detection or evaluation results. A detection protocol that applies active middleware components to share detection information is our focus in designing a communication method in Co-net.

c. *How can CE measurement be defined to represent the relative seriousness of a given CE event?*

   When conflicts and errors happen, the methods to measure the severity of a problem are a relevant question. If the CE seriousness measure can be defined properly, this measure can provide the Co-U with the CE resolution priority when performing the next-stage resolution process. Besides, determining that a Co-U has high frequencies of CE and can cause potentially serious damage implies that this Co-U's tasks have to be redesigned to prevent damages; its tasks can be quarantined until its performance is improved.

The paper is organized as follows: Section 2 addresses the research background and literature review. In Section 3, conflicts and errors in a collaborative environment are defined. Based on this definition, we model CE propagation. Section 4 describes a proposed conflict/error detection protocol (CEDP) and conflict/error detection agent (CEDA) that focuses on task specification conflicts and CE propagation. In Section 5, an experiment that applies the developed CEDP in three basic dependency networks is conducted. Finally, Section 6 includes observations, conclusions and recommendations.

2. Research Definitions and Literature Review

Cooperation unit (Co-U): the cooperation unit (Co-U) is an autonomous working unit that performs tasks to achieve its goals and coordinates with other Co-Us to accomplish the common goal of a set of Co-Us. They can be humans, machines, workstations, enterprises, etc., depending on the scope of the collaborative environment. A Co-U can be represented by five tuple: *G*, *T*, *A*, *R*, and *S* (Eq. 1):

$$Co-U = \{G, T, A, R, S\} \tag{1}$$

Where *G*: Co-U's Goal, e.g., profit and/or survival;
T: Set of Tasks that a Co-U must perform to achieve its goal (*G*);
A: Set of Activities that a Co-U runs in order to accomplish the tasks (*T*);
R: Set of Resources that a Co-U consumes while running the activities (*A*);
S: Set of States that represents a Co-U's current situation of *G*, *T*, *A*, and *R*.

Each Co-U performs the tasks (*T*) by executing the activities (*A*) and using the limited resources (*R*) to achieve the goal (*G*). The state (*S*) is used to describe the current situation of goals (*G*), tasks (*T*), activities (*A*), and resources (*R*). The goal of a Co-U can be defined by it or assigned by its supervisor. A Co-U might receive tasks from other Co-Us, assign tasks to other Co-Us, or define its tasks for fulfilling its local goals. The activities are practical actions or procedures that are needed to complete tasks. In order to execute activities, each Co-U must consume its own resources or share resources.

Coordination Network (Co-net): Co-net is a network that enables a collaborating process among a group of Co-Us such as agents, enterprises, and operators to achieve a common goal. Each Co-U within the Co-net exchanges information to achieve its individual tasks (*T*) and fulfill the common goal of all members. We extend Anussornnitisarn's (2003) definition of Co-net by adding a common-goal and resource component:

$$\text{Co-net} = \{\pi, \upsilon, \tau, \alpha, \lambda, \sigma\} \tag{2}$$

where π: Set of autonomous Co-Us in a Co-net ($Co-U \in \pi$);
υ: Set of common goals that should be achieved; G is a subset of υ. ($G \in \upsilon$);
τ: Set of tasks that can be processed in a Co-net; *T* is a subset of τ. ($T \in \tau$);
α: Set of activities for each participant (π); *A* is a subset of α. ($A \in \alpha$);
λ: Set of resources that can be used in Co-net; *R* is a subset of α. ($R \in \lambda$);
σ: Set of coordination mechanisms that are used by each Co-U (π).

The Co-net is a complete view of an operation working space. It might be an assembly line, workflow system, supply system, design teamwork or any kind of collaborative environment. Many elements comprise a Co-net, for instance, people, devices, resources, capital, social negotiation, information systems, etc. All elements of a Co-U such as goals, tasks, activities, and resources are subsets of a Co-net's components, respectively.

The Co-U and Co-net define the scope of a collaborative environment. A Co-U represents a participant of collaborative activities. A Co-net is a combination of many Co-Us. Each Co-net has its own goal (common goal) that should be achieved by involving Co-Us. Every Co-U communicates with others by certain coordination mechanisms. The tasks, activities, resources of a Co-net are preformed, executed, and shared among Co-Us, respectively.

Task Dependency

In a distributed collaborative environment, the coordination between autonomous participants is about managing task dependencies [Anussornnitisarn and Nof, 2001, 2003]. Based on Malone et al. and Khanna et al.'s work, three essential task dependencies that are flow, merging, and sharing represent the base of any type of complete task dependencies. Combining basic types of task dependencies, a complete task flow or Co-net can be formed and the relationships of Co-Us belonging to the Co-net can be described by these task dependencies.

Since each collaborative member has only local information about the task dependencies, if conflicts or errors occur in a Co-net, the lack of CE information might cause wrong or inadequate decisions in the resolution processes. In either case, the conflict or error propagates within the Co-net. In this research, we study the conflict or error propagation in three structures of Co-nets: linear, tree, and parallel, which represent the most common types of task dependencies.

a. Linear Co-net

Linear Co-net is a relatively simple Co-net (see Figure 1 a). In this Co-net, each Co-U receives tasks from only one Co-U and offers tasks to only one Co-U; it can represent a sequential supply system or an assembly line. Because a linear Co-net has a relatively simple, predefined linear relation between Co-Us, the analysis evaluation of conflicts and error detection is relatively simple.

b. Tree Co-net

Tree Co-net can be treated as a combination of several linear Co-nets. In a tree Co-net, each Co-U offers tasks to more than one Co-U, therefore, tasks are diffused out (see Figure 1 b). In the distribution or delivery industry, this kind of Co-net is common. For example, a supplier delivers finished products to its costumers.

c. Parallel Co-net

Parallel Co-net is a relatively complicated Co-net. Each Co-U not only offers tasks to more than one Co-U, but can also receive tasks from more than one Co-U, thus, tasks are diffused out and merged (see Figure 1 c). This kind of Co-net represents more complex collaboration environments, for example, a supply-network, an assembly line, design project teamwork, and so on.

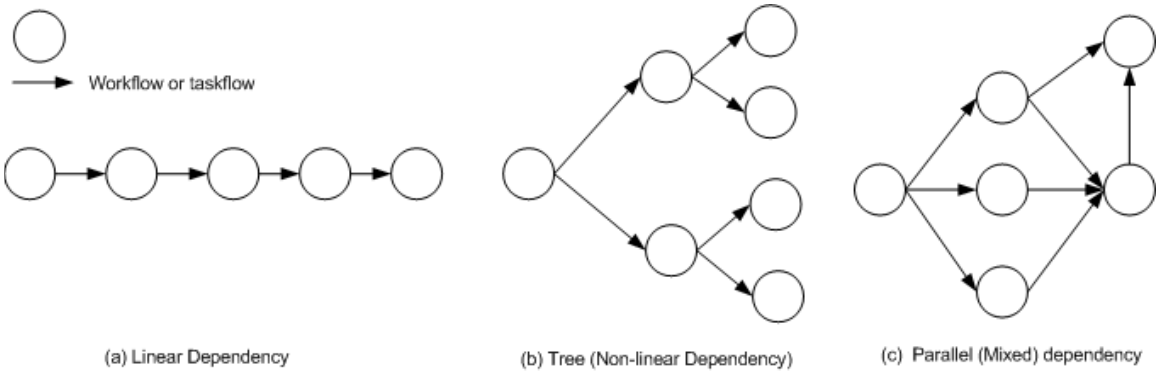


Figure 1. Linear (a), tree (b), and parallel (c) task dependencies

Protocols and Agents in Active Middleware

Based on Bakken's (2003) definition in the Encyclopedia of Distributed Computing, Middleware is a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is an enabling layer of software that resides between the business application, and the networked layer of heterogeneous platforms and protocols [Anussornnitisarn, 2003]. According to Anussornnitisarn and Nof [2001, 2003], the major components of active middleware are: Multi-Agent Based Systems (MAS), Workflow Management Systems (WFMS), coordination protocols, Decision Support Systems (DSS), modeling language/tools, task/ activity databases (or knowledge base). Their conceptual model is shown in Figure 2.

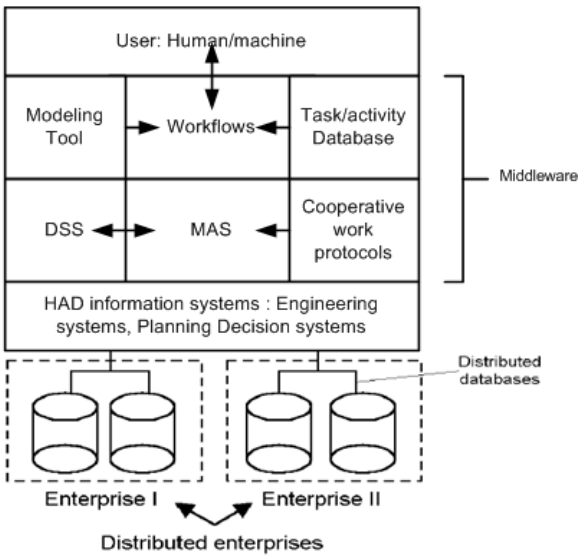


Figure 2. Active middleware architecture [Anussornnitisarn et al., 2001]

These six components can be classified in two categories: active components and support components. Active components include Multi-Agent Based Systems (MAS), WorkFlow Management Systems (WFMS), and the coordination protocols. Each protocol comprises a set of mechanisms acting or reacting according to a changing environment. Support components include Decision Support Systems (DSS), modeling tools, and task/activity databases, which respond to requests from the active components.

In a distributed, collaborative environment, each participant has its own goals, tasks, and resources. The management system of each participant, e.g., information system, project planning system, database system, Enterprise Resource Planning (ERP) system, or manufacturing execution system plays an important role in managing the project, manipulating the operation of tasks and monitoring the resources. Because of the heterogeneity of those distributed participants' systems, smooth collaboration between these systems is difficult to achieve. Active middleware serves as a bridge between distributed systems and provides an integrated platform to communicate and cooperate. In this research, the conflict/error detection model will be defined and evaluated upon these active middleware components.

Exception Handling

Workflow Management Systems (WFMSs) are increasingly being deployed to deliver e-business transactions across organizational boundaries. To ensure high service quality in transactions, exception-handling schemes are needed. Klein and Dellarocas (2000) define exception as "any deviation from an ideal collaborative process that used the available resources to achieve the task requirement in an optimal way". An exception can thus include errors in performing a task or communicating results between agents, deficient response to changes in tasks or resources, missed opportunities, and so on. Because of anticipated or unanticipated situations, the deviations (exceptions) of those workflow processes from their specifications are unavoidable. In our work, we treat an exception as a combination of error and conflict.

Klein and Dellarocas proposed a hierarchy taxonomy business process repository to identify the exception in a workflow system [Klein 2000, Klein and

Dellarocas 2000]. It involved the development and evaluation of workflow for handling the multi-agent exception that can occur in a workflow environment. For each exception type, the workflow designer can then decide how to extend the workflow models to prepare for anticipating or detecting the exception. This concept can improve the exception handling quality. In this research, a knowledge-based CE repository can be applied to store information of CE and be accessed by participants.

In summary, several concepts and contributions delivered by researchers in various areas provide inspiration in designing our detection model. Active middleware conceptual models, agent technology applied in distributed collaboration, knowledge-based repository for improving CE detection and further resolution process are useful developments from reviewing these research activities.

3. CE Definition and CE Propagation

Many words or terms are used to describe problems in a collaborative environment such as error, failure, mistake, defeat, exception, conflict, misunderstanding and so on. Since they can be unclear and ambiguous, we define error and conflict from a "boundary" point of view, and consider an important phenomenon, CE propagation.

3.1 Definition of Error

An error is an unintentional or unexpected event that prevents a task performer from achieving the goal of a process (Table 1). In this research, we define an error as follows:

Error: The difference between the current state and the expected state, with the difference being greater than the tolerance, after operations by a Co-U.

Based on this description, a mathematical representation of an error can be defined as:

$$\text{Error} = |S(t)^* - S(t)'| > \text{Tolerance} \quad (3)$$

Where $S(t)^*$: Current state at time t;
$S(t)'$: Expected state at time t;
–: An applicable difference operator;
Tolerance: The tolerance of allowing a state difference.

The current state indicates the practical output or current condition of a Co-U. The expected state is a planned output or objective that should be obtained by a Co-U's operation. If the difference between the current state and expected state exists in time t, and that difference is greater than the tolerance, there must be an error in the operation. In this equation, we focus on the state that can be quantified. Based on the previous definition, we can make some assumptions regarding errors:

a. An error can happen unexpectedly and suddenly.
b. It is not a normal situation.
c. There are different error levels from serious to slight.

3.2 Definition of Conflict

When people or organizations work autonomously and have to merge or exchange their results together, a conflict might emerge. Also, the different resource management of different organizations might lead to a resource conflict if organizations share a common resource (Table 1). In this research, we define conflict as:

An inconsistency between Co-Us' goals, dependent tasks, associated activities, or plans of sharing resources, after coordinated by cooperative Co-Us.

Based on this description, a mathematical representation can be defined as

$$Conflict = \{[G_i \rightarrow\leftarrow G_j] \vee [T_i \rightarrow\leftarrow T_j] \vee [A_i \rightarrow\leftarrow A_j] \vee [R_i \rightarrow\leftarrow R_j]\} \quad (4)$$

where $\rightarrow\leftarrow$: Operator denotes "in conflict with"

i and *j*: Indexes of different Co-Us
G_i, G_j : Goals of Co-U *i* and Co-U *j*
T_i, T_j : Dependent tasks of Co-U *i* and Co-U *j*
A_i, A_j : Associated activities of Co-U *i* and Co-U *j*
R_i, R_j : Sharing resources of Co-U *i* and Co-U *j*

Some assumptions regarding conflict that we hold in this research:

a. A conflict occurs among more than one subsystem or cooperative members.
b. A conflict can be detected or solved by coordination or negotiation between involved members.
c. Inconsistence is a major reason of conflict.

Table 1. Examples of error and conflict

Causes of error	Causes of conflict
1. Task misunderstanding	1. Time (schedule) mismatch
2. Facility crash, machine fatigue	2. Unexpected cost, profit
3. Unaccepted quality	3. Different design concept
4. Abnormal event, human mistake	4. Different process or operations
5. Exceed resource capability	5. Various task specification
6. Insufficient ability	6. Resource overuse
7. Error in specification	7. Violation of common goal
8. Lack of material, technologies	8. Collision (path conflict)
9. No matched manufacturing process	9. Different format
10. Network break down, bugs	10. Different units of measure

3.3 Comparison of Conflict and Error

In this section, we define a Co-U's system boundary and graphically locate the occurrence of a conflict or an error. The position of a CE event differentiates between a conflict and an error and addresses CE propagation in a collaborative environment.

A boundary of a Co-U's system identifies a Co-U in a Co-net. Inside the boundary of a Co-U, major elements of a Co-U such as goals, activities, tasks and resources can be located. When elements are located inside a Co-U's boundary, it means that they belong to this Co-U (Figure 3).

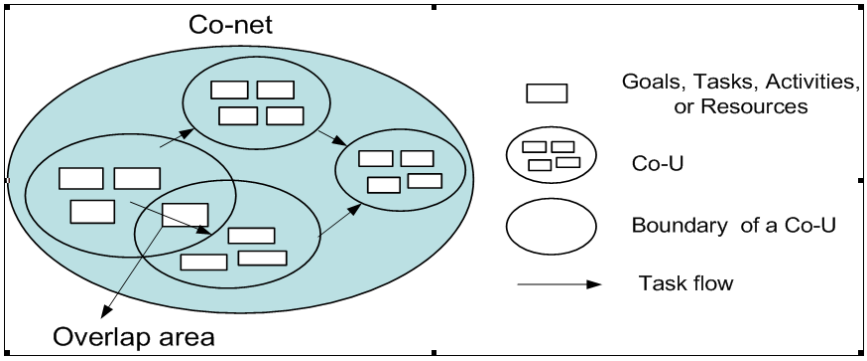


Figure 3. Boundaries of a Co-U and a Co-net

Based on the definition above, the scopes of an error and a conflict can be defined. The scope of an error is within a Co-U's boundary but not inside any overlap area of other Co-Us. For instance, a mistake or failure might occur when a Co-U executes its activities. If these Co-U activities are not correlated with other Co-Us, the cause of the mistake or failure should be directly related to that Co-U and other Co-Us will not share responsibility. In this scenario, we define a mistake or failure as an error. In Figure 4, a "star" illustrates an error event.

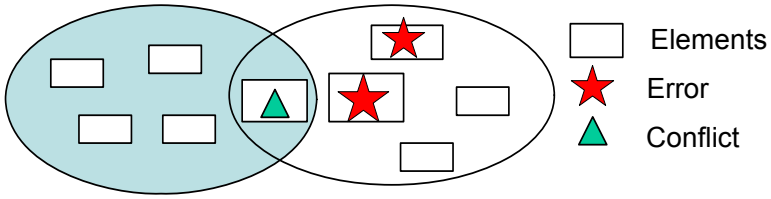


Figure 4. Scope of conflict and error

On the other hand, if an unexpected event is associated with other Co-Us' interventions, e.g., performing activities together or occupying shared resources; we define this kind of mistake or failure as a conflict. Since a conflict must occur because of an interaction between Co-Us, the scope of the conflict is in the overlap area of different Co-Us. We use a "triangle" in Figure 4 to illustrate a conflict event. Table 2 shows a comparison between a conflict and an error based on the boundary definition.

Table 2. Comparison of scope between a conflict and an error

Error	Conflict
1. Occurs inside a Co-U's boundary but not in any overlap area with other Co-Us 2. Caused by the Co-U's mistakes or misunderstandings	1. Happens within an overlap area of a group of Co-Us 2. Caused by the interaction between Co-Us

3.4 Conflict / Error Propagation

Once an error occurs inside a Co-U's boundary, it might cause an error chain-effect in this Co-U, meaning that errors might also lead to other errors (derivative errors). For example, in a workstation of an assembly line, the wrong geometric dimension of one component caused by negligence might also affect operations in other workstations. This kind of situation is common not only in manufacturing but also in business and other collaboration environments. We define this kind of situation as *error propagation*.

In addition, an error inside a Co-U's boundary might cause a conflict because of task interdependency or common resources shared by several Co-Us. The task dependency can be treated as the interaction between Co-Us when they cooperate to achieve a common goal. Combining several task dependencies or sharing common resources can establish a Co-net. If the task dependency becomes complicated, CE propagation might occur in a Co-net because of the complex correlation between Co-Us. For example, if one product supplier cannot fulfill the task request on time because of an internal operation error, this error event will also affect its customers' purchasing plan, and cause a conflict situation. In this case, the delivery schedule conflict propagates out and affects more members if this CE problem is not resolved immediately. This kind of situation is defined as *conflict propagation* (Figure 5).

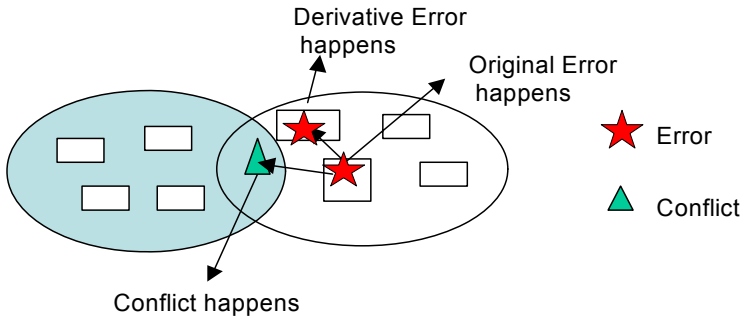


Figure 5. Conflict / error propagation

The conflict propagation in this research focuses on the CE problems broadcast by circumstances. If a CE handler cannot solve a CE problem correctly or consider correlated Co-Us' situations, the conflict problem propagates.

4. Methodology

In this section, the conflict/error detection protocol (CEDP) is proposed to perform CE detection in distributed Co-nets. The CEDP is used to detect CE events after a task has been assigned in a collaborative environment. Each Co-U in a Co-net is equipped with a software agent called conflict/error detection agent (CEDA) to perform CE detection and collect information. CEDAs' functions are to monitor and evaluate a Co-U's task processing activity. Collected detection information is shared and transmitted by CEDP. Active middleware components such as a task database, a CE knowledge base, and a DSS, can help support a CEDA and the CEDP with decision making and knowledge gathering. The main components of CEDP are illustrated in Figure 6.

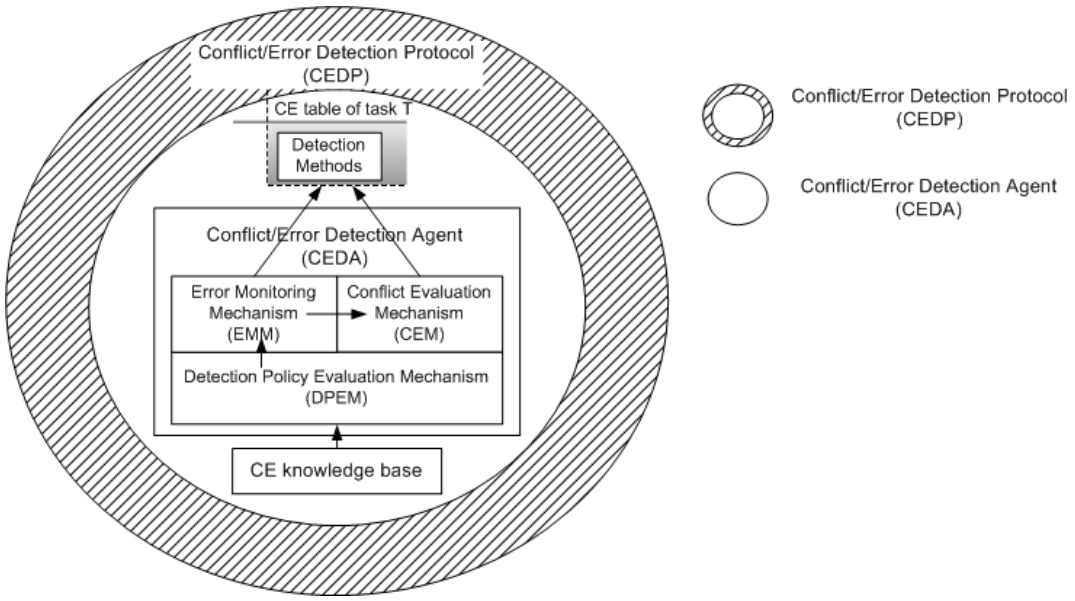


Figure 6. Components of Conflict / Error Detection Protocol (CEDP)

4.1 Conflict / Error Detection Agent (CEDA)

The Conflict/Error Detection Agent (CEDA) is a software agent that is used to perform the CE detection process in a Co-U. There are three components in the CEDA: 1. Detection Policy Evaluation Mechanism (DPEM), 2. Error Monitoring Mechanism (EMM), and 3. Conflict Evaluation Mechanism (CEM). The three of them coordinate together to detect possible errors and conflicts and notify the correlated (potentially infected) Co-Us by CEDP. Figure 7 shows the components of CEDA with their input and output.

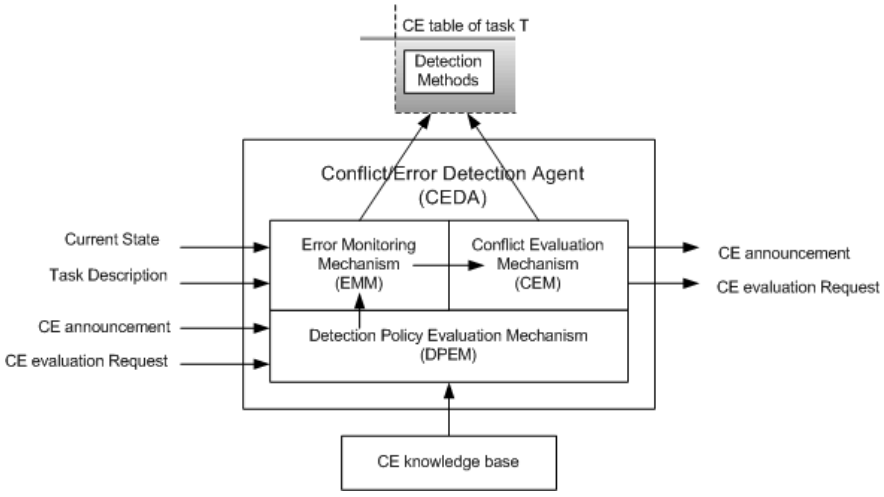


Figure 7. Input, output and components of CEDA

Detection Policy Evaluation Mechanism (DPEM)

The Detection Policy Evaluation Mechanism (DPEM) is responsible for evaluating each detection method (*dM*) regarding each possible CE problem and generating the detection policy $(dP)_T$ for performing a task *T*. The detection policy is a guideline of how to detect a particular CE problem. Based on the information stored in

the CE knowledge base, DPEM can evaluate all applicable *dM*s for a specific CE and select the best one if only one *dM* is required. Then, DPEM will evaluate the cost-effectiveness of the selected *dM* and decide whether it should be applied based on cost. After evaluation, DPEM implements the effective detection policy $(dP)_T$ in executing the CE detection when the Co-U intends to perform task *T*.

Error Monitoring Mechanism (EMM)

The error monitoring mechanism (EMM) of the CEDA is in charge of continually monitoring the Co-U's activities. CEDA obtains the run-time current state (*Θ*) from the input of different monitoring devices. Based on the Co-U's plan of resource policy, task specification, and current state, an error can be detected and recognized. Until the error situation is recovered, EMM will continue to trace this error.

Conflict Evaluation Mechanism (CEM)

The conflict evaluation mechanism (CEM) is a kernel of the CEDA with two functions, calculation and comparison, to detect possible conflicts when a Co-U performs a task. Through these two functions, CEDA can standardize all current states (*Θ*), tasks constrains (*Ω*), and compare them to find a conflict when unsatisfied task constraints (*Ω'*) exist. Figure 8 shows a complete perspective of a CEM.

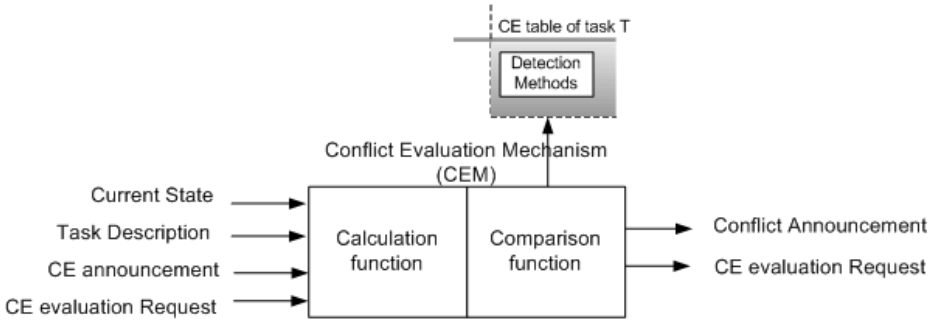


Figure 8. Conflict Evaluation Mechanism (CEM)

Conflict-severity (*Δ*)

A *conflict-severity* (*Δ*) is used to measure the degree of conflict. If unsatisfied task constraints (*Ω'*) exist, CEDA applies this *Δ* measurement by summing all weights of unsatisfied task constraints (*Ω'*). If *Δ* is high, it means that more task constraints cannot be satisfied and the conflict of task processing is at a higher level of severity.

$$\text{Conflict-severity } (\Delta) = \sum_{i=1}^{|\Omega'|} w_i' \qquad (5)$$

where $|\Omega'|$ is the total number of unsatisfied task constraints;

w_i' is the weight of i^{th} unsatisfied task constraint of a Co-U.

Once a CEDA agent receives a CE announcement or CE evaluation request from its cooperative Co-Us, it will calculate *Δ*. If all task constraints of this agent can be satisfied, *Δ* equals 0, meaning no conflict occurred based on that updated task proposal.

Conflict / Error Knowledge Base

The CE knowledge base is a repository of CE detection information that can be accessed by the CEDA of the Co-Us. A CE table is defined and maintained to store all

information about the CE problems in the knowledge base. The CE table stores information such as the CE index (x), CE description, CE occurrence frequency (F_x), CE detection method (dM_{xy}), detection accuracy (dA_{xy}), average detection cost (dC_{xy}), detection cycle time (dT_{xy}), possible damage (D_x), and potential CE resolution. The structure of a CE table and an example are presented in Table 3.

The information of detection methods stored in a CE table provides the CEDA with sufficient data to analyze or examine possible detection procedures faced with an analogous CE problem.

Table 3. The CE table of a task *T*

Index of Co-U	
CE index (x)	The unique index of a CE problem
CE description	The description of this CE
CE occurrence frequency (F_x)	F_x is the occurrence frequency of a CE event. It is the number of occurred CE problems per time unit. For example, F_x might be 0.01/second. Higher frequency means this CE will happen more often
Damage (D_x)	The estimate or possible damage of a CE problem
Detection methods (dM_x) Detection method (dM_{xy}) $\forall dM_{xy} : dM_{xy} \in dM_x$	dM_x is a set of applicable detection methods for CE x dM_{xy} is a detection method y to detect CE x This element stores a set of applicable detection methods that can be used to detect CE x. There could be more than one detection method that can be used to detect a CE problem. y is the index of the detection method (y >= 1).
Detection accuracy (dA_{xy})	The accuracy of a specific detection method y, e.g., detection rate of sensor
Average detection cost (dC_{xy})	dC_{xy} is the average detection cost per time unit of the specific detection method y to detect CE problem x. For instance, dC_{xy} might be \$10/second or \$600/minute. Each detection method might have a different detection cost
Detection cycle time (dT_{xy})	dT_{xy} is the time that is spent on finishing a detection cycle of the specific detection method y. For example, in manufacturing, a visual sensor might need 2 seconds to finish inspection. In this case, the dT is 2 seconds. Each detection method might have a different detection time
Potential CE resolution	This element describes the possible CE resolutions

4.2 Conflict / Error Detection Protocol (CEDP)

The proposed conflict/error detection protocol (CEDP) is an agent-based protocol that facilitates the exchange of detection information between Co-Us in a Co-net. CEDP enables CEDA to send or receive CE announcements, CE evaluation requests, and CE evaluation results. Through this protocol, not only CE events can be transmitted within a Co-net, but also the CE evaluation information can be shared when it is needed for the detection process.

Message Definition

CEDP handles the detection information exchange process among Co-Us. Three kinds of messages are transmitted by this protocol: a CE announcement, a CE evaluation request, and a CE evaluation result. We define π^* to represent a set of Co-Us which send out CE announcements or evaluation requests, and $\pi^{\#}$ to represent a set of Co-Us which receive them.

a. ***CE announcement***: Once a Co-U detects a CE inside its boundary, CEDA will broadcast the CE announcement to other correlated Co-Us. Since Co-Us collaborate to achieve a common goal, each cooperative Co-U is responsible to warn other Co-Us of the CE. When a CE announcement is received, the CEDA of a Co-U will retrieve the CE table for the CE announcement and evaluate this CE event.

b. ***CE evaluation request***: A CE evaluation request is a message to ask a Co-U to evaluate its activity and return its evaluation result. A CEDA can request evaluations from cooperative Co-Us and acquire CE evaluation information for understanding the influence of a task specification change. The evaluation request contains task specification of a modified task. The request receiver can examine its task processes and evaluate their potential influence.

c. ***CE evaluation result***: After a CEDA receives a CE announcement or CE evaluation request, the CEM of the CEDA begins to evaluate its task activities and return evaluation results to the requesters. If a CE is detected in the evaluation process, the information regarding the detected CE (index of CE and CE table) and non-zero conflict-severity value will be included in the evaluation result. Otherwise, a zero conflict-severity value is returned with the evaluation result.

CEDP Operation

Two kinds of CEDP operations are executed between Co-Us. One occurs when a CE is detected; another when a task specification is changed.

a. A CE is detected

Once a Co-U detects a CE problem that occurs within its boundary, this Co-U is responsible to inform other correlated Co-Us. Every Co-U that receives a CE announcement evaluates the potential influence of this CE and sends back the evaluation result. Then, any Co-U that detects a CE, can also estimate the potential influence on other Co-Us.

Illustration: In Figure 9, a CE occurs and is detected by Co-U3. Co-U3 then sends out CE announcements to its correlated partners: Co-U1, Co-U2, Co-U4, and Co-U5. Each CE announcement receiver evaluates the detected CE event and returns the CE evaluation result to Co-U3. The detailed operation is listed in Table 4.

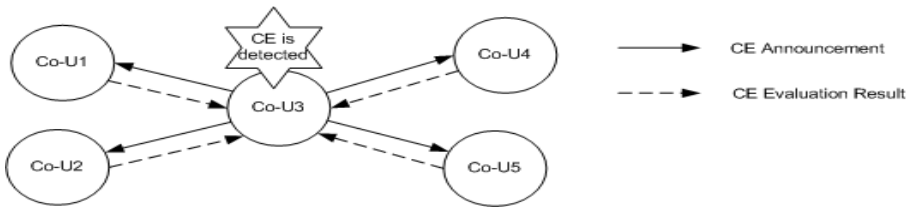


Figure 9. CEDP operation when a CE is detected inside a Co-U's boundary

Table 4. Conflict / error detection protocol when a CE occurs

Conflict / Error Detection Protocol	
Co-U (π^*)	Co-U($\pi^{\#}$) ($\|\pi^{\#}\|$>= 1)
Variables: CE announcement (*a*) 1. Index of sender Co-U (π^*) 2. Index of receiver Co-U ($\pi^{\#}$) 3. Time 4. Conflict-severity (Δ) 5. Index of CE (*x*) 6. The CE table Task constraints (Ω)	*Variables:* Evaluation result (*e*) 1. Index of sender Co-U ($\pi^{\#}$) 2. Index of receiver Co-U (π^*) 3. Time 4. Received CE announcement (*a*) 5. Conflict-severity (Δ) 6. Index of CE (if CE is detected) 7. The CE table (if CE is detected) Task constraints (Ω)

Statements:
A CE happens
Update CE table(*x*) in CE knowledge base
Initiator()
While{
(1) Evaluating current task constraints (Ω)
(2) Calculating the conflict-severity (Δ)
(3) π^* -> $\pi^{\#}$: Sending (CE announcement)
(4) Wait()
 	
(9) Receiving (evaluation result)
(10) Updating received conflict-severity
}All cooperative CEDA finish communication
	Statements:
	
	Responder()
	{
	
	(5) Receiving (CE announcement)
	(6) Evaluating task constraints (Ω) with received CE announcement (*a*)
	(7) Calculating conflict-severity (Δ)
	If (CE) then {
	Calculating conflict-severity (Δ > 0) }
	Else{
	Conflict-severity = 0 }
	(8) π -> π^* : Sending (evaluation result)
	
	Recursive CE detection
	If (Δ > 0) {
	Intitiator()
	While{
	π' -> π : Sending (CE announcement)
	} All cooperative CEDAs finish communication

b. A task specification is changed

If a Co-U wants to change the task specification, this change might affect other partners and cause conflicts (Figure 10). Figure 10 (A) shows a task dependency of a Co-net that includes four members. Figure 10 (B) indicates a task specification change in Co-U3. The Co-U3 will send an evaluation request with its changed task specification to cooperative partners: Co-U1, Co-U2, and Co-U4. After evaluating the task activities with the new task specification, all receivers (Co-U1, Co-U2, and Co-U4) will send back the evaluation result. Table 5 describes the details of CEDP operation when a task specification is changed.

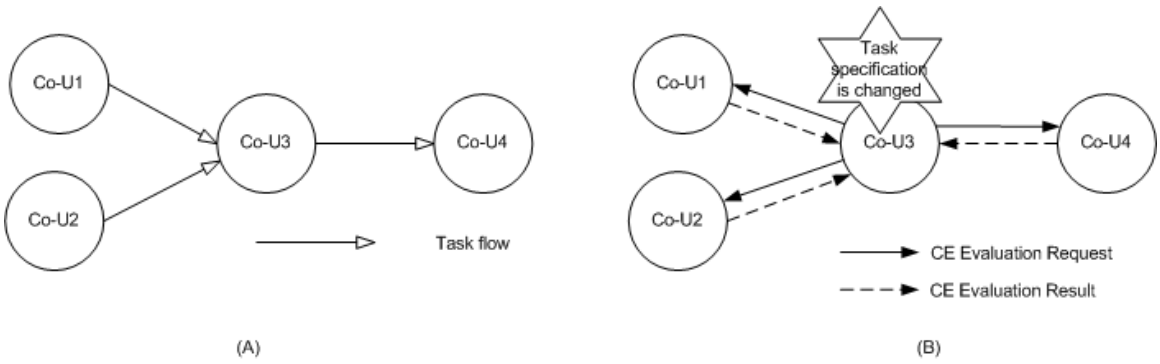


Figure 10. CEDP operation when a task specification is changed

Table 5. Conflict / error detection protocol (CEDP) when a task specification is changed

Conflict / Error Detection Protocol	
Co-U (π^*)	Co-U ($\pi^{\#}$) ($\|\pi^{\#}\|$>= 1)
Variables: CE evaluation request 4. index of sender Co-U (π^*) 5. index of receiver Co-U ($\pi^{\#}\pi$) 6. time 7. modified task specification	*Variables:* Evaluation result (e) 1. index of sender Co-U ($\pi^{\#}$) 2. index of receiver Co-U (π^*) 3. time 4. received CE announcement (a) 5. conflict-severity (Δ) 6. index of CE (if CE is detected) 7. CE table (if CE is detected) Task constraints (Ω)

Statements:
 	
A task specification is changed
Initiator()
While{
(1) π^* -> $\pi^{\#}$: Sending (CE evaluation request)
(2) wait
 	
(7) Receiving the evaluation result (e)
(8) Updating received conflict-severity
}
}All cooperative CEDAs finish communication
	Statements:
	
	Responder()
	{
	
	(3) Receiving (evaluation request)
	(4) Evaluating task constraints (Ω) with updated task specification
	(5) Calculating conflict-severity (Δ)
	If (CE) then {
	Calculating conflict-severity (Δ > 0) }
	Else{
	Conflict-severity = 0 }
	(6) π -> π^* : Send(evaluation result)
	
	Recursive CE detection
	If (Δ > 0) {
	Intitiator()
	While{
	π' -> π : Sending (CE announcement)
	……
	} All cooperative CEDAs finish communication

4.3 CEDP with Active Middleware Components

In this research, CEDA and CEDP can be treated as an agent middleware based on the taxonomy of middleware [Bishop and Karne, 2003]. CEDA is a software entity located in each Co-U and it is in charge of monitoring and evaluating the current activities. CEDP provides a communication platform for all Co-Us in a Co-net. CEDA can notify other Co-Us of detected CE events and share detection information with others through CEDP.

Active middleware components defined in Anussornnitisarn's work can be supportive elements for CEDP (Figure 11). Decision support systems (DSS) provide a CE detection reasoning when each CEDA receives detection information from its monitoring mechanism and other cooperative Co-Us. A DSS can support the evaluation and examination processes of a CEDA to identify CE problems.

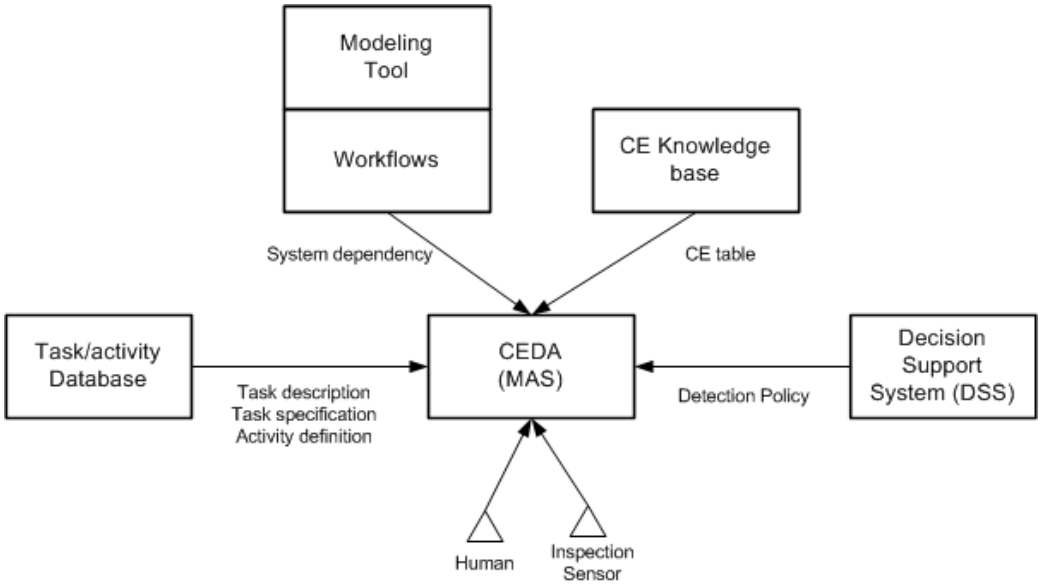


Figure 11. CEDA with Active middleware components

A task/activity database (or knowledge base) contains detailed information about tasks and activities related or derived from a Co-U's goal. The descriptions of each Co-U's goal, tasks, activities, and resources are stored in this knowledge base. Accumulating these descriptions and CE problem descriptions, knowledge or experiences about CE detections can provide Co-Us with sufficient information for a further resolution stage.

Workflow Management Systems (WFMS) provides a workflow structure for a Co-net. Each CEDA follows this structure to identify its communicating predecessors and successors among Co-Us. No matter what kind of task dependency among Co-Us exists, each CEDA of a Co-U can exchange detection information with a connected Co-U.

In terms of function, a CEDP with a CEDA is a two-way software middleware among Co-Us and between active middleware with each Co-U's applications. On the one hand, through a CEDP operation, a CEDA communicates with other CEDAs to share detection information. On the other hand, a CEDA acquires detection information from Co-U's activities and stores it in an active middleware knowledge base. Next, active middleware components such as DSS help support a CEDA to make a decision on

detection policy. Therefore, a CEDP constitutes a two-way middleware CE detection model (Figure 12).

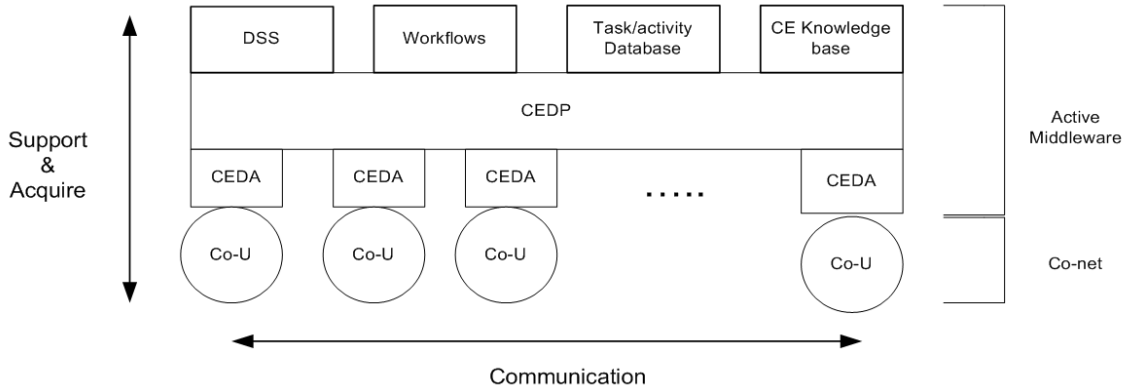


Figure 12. CEDP is a two-way middleware in CE detection

5. Experiments and Results

A set of experiments has been designed and conducted with a prototype CEDP. The objective of these experiments is to implement a CEDP, apply the conflict-severity (Δ) to measure the potential implications and seriousness of a CE problem, recognize the CE-infected Co-U (π') by using CEDP, and understand how to design a better CEDP. We utilize the conflict-severity (Δ) and the CEDP to evaluate the CE problems in three basic forms of task dependence Co-nets: linear, tree, and parallel.

5.1 Experimental background

Before describing the experiments, we first address some assumptions and background information.

a. No conflict or error is evident in the Co-net network establishing stage: During the Co-net network establishing stage, we assume for simplicity that no conflict or error happens. When a Co-net is established by assigning tasks, every Co-U must agree with the initial task dependencies. In this experiment, this stage is called “initial stage of a Co-net”. In an ideal situation, each Co-U coordinates with other Co-Us to achieve its own goal and a common goal from its initial stage. However, that “ideal” situation is rare because errors and conflicts are unavoidable in a Co-net and can happen at any time.

b. Conflicts can propagate to other cooperative members: Contrary to the initial stage of a Co-net, the CE propagation stage of a Co-net means that an unexpected event occurs and can propagate to other Co-Us after the initial stage. Since CE propagation is an important phenomenon in any collaboration environment, this experiment is designed to search CE-infected Co-Us (π') in a Co-net by monitoring and evaluating the conflict-severity (Δ).

In the experiments, the Co-net environment has to be formed before the CE detection. Experiments #1, #2 and #3 establish Co-nets based on linear, tree, and parallel dependency networks, respectively. Each Co-U in a network can use a CEDP to communicate with other Co-Us and share information about CE problems.

5.2 Measures of Conflict / Error Detection Protocol (CEDP)

Three measures, which are the number of CE-infected Co-Us (π'), conflict-severity (Δ), and total communication time (T_c) are used to measure the performance of a CEDP in the three forms of Co-nets.

a. **Total communication time** (T_c) represents the accumulating time for each Co-U to communicate with others. Once a CE happens in a Co-U, a CEDA will communicate with collaborative partners and exchange the conflict information until all CEDAs finish their evaluations.

$$T_c = \sum_{j}^{|COM|} C\_time_j \tag{6}$$

where T_c is the total communication time;

j is the index of communication activity;

|*COM*| is the total number of communication activities;

C_time_j is time spent on communication activity *j* performed by a CEDA.

b. The number of CE-infected Co-Us (|π'|) represents the total number of Co-Us that detect a CE after evaluating their current task constraints (Ω) and CE announcements.
c. Conflict-severity (Δ) defined as Equation (6) represents the seriousness of conflict events. It accumulates the weights (*w*) of the unsatisfied task constraints (Ω').

5.3 Experimental Procedures

Based on the design of the CEDP, each CEDA monitors the Co-U's activities by examining its current states (Θ) and task constraints (Ω) to detect a CE. The experiment procedures are divided into two stages: the initial stage and the CE propagation detection stage (Figures 13 and 14).

Initial stage

a. By using Excel with VBA, Visual Basic language initially constructs three task activity dependence Co-nets (linear, tree and parallel) with the same number of Co-Us.
b. Randomly assign an error event for a Co-U. Then, this Co-U is the initial CE-infected Co-U (π').
c. The initial CE-infected Co-U (π') evaluates the CE problem and calculates the conflict-severity (Δ).
d. The initial CE-infected Co-U (π') checks for possible task specification modification to locally solve the CE problem.
e. The initial CE-infected Co-U (π') sends out the CE announcement with that task specification modification to the connected Co-Us once a CE is detected.

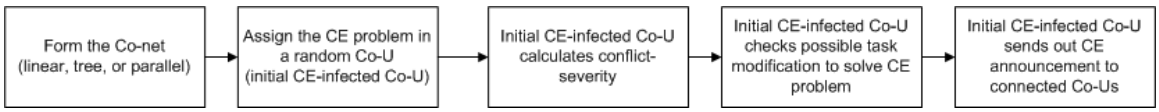


Figure 13. Experimental procedure – initial stage

CE propagation detection stage

a. Once a Co-U, ($\pi^{\#}$), receives CE announcements, it will check whether the detected CE in the initial CE-infected Co-U (π') and task specification modifications affect its task activities.
b. Next, $\pi^{\#}$ calculates the conflict-severity (Δ). If no conflict, Δ is equal to 0. If this $\pi^{\#}$ is affected by more than one Co-U, its Δ will sum all weights of unsatisfied task constraints. In this case, Δ could be greater than 1.
c. Finally, $\pi^{\#}$ sends back the evaluation result.
d. If a new CE is detected, recursively detect a possible CE problem in a Co-net
    a) First, $\pi^{\#}$ checks the possible task specification modifications that can solve the detected CE problem locally.
    b) Next, $\pi^{\#}$ sends out the CE announcement with that task specification modification to connected Co-Us.

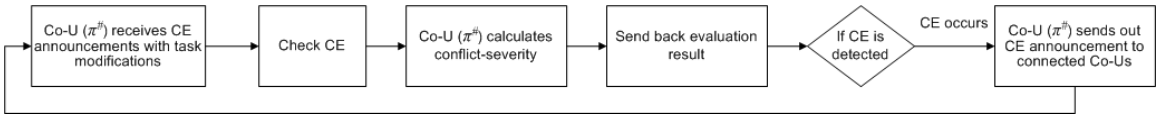


Figure 14. Experimental procedure – CE propagation detection stage

By changing current states (Θ'), the experiment can attribute a CE problem to a particular Co-U. This Co-U compares task constraints (Ω) and changed current states (Θ'). If $\Omega > \Theta'$, a CE has occurred at this Co-U and has been detected. However, how to change the current states (Θ') remains open. To address this situation in this experiment, we define a CE control factor (ε) to control the relative changing range of current states (Θ'). The CE control factor (ε) is a fraction between 0 and 1. When ε is larger (range of Θ' is also large), the relative change in current states (Θ') will be smaller. It also means that the conflict event is more likely to occur in this experiment ($\Omega > \Theta'$). The range of Θ' is defined below as:

$$\text{Range of } \Theta' = [\Theta - \varepsilon, \Theta] \quad (6)$$

where Θ' is the changing current states;
Θ is the original current states;
ε is the CE control factor.

In this experiment, the parameter ε is applied to control the emergence of conflict situation; if ε is high, the conflict most likely happened in a given Co-net ($\Omega > \Theta'$).

Visual Basic for Applications (VBA), Excel's programming language, is used to develop the CEDP's logic [Albright, 2001]. In the initial stage, the VBA program creates a Co-net with 20 Co-Us and assigns the initial values to each Co-U's data table. After finishing the data initialization, a CE problem is attributed randomly for a Co-U. Then, the CE propagation detection begins to search all CE-infected Co-Us (π'). The experimental parameters, input and output of this simulation environment are shown in

Table 6. Regardless of the form of task dependency (linear, tree, parallel) of a Co-net, each experiment follows the same settings.

Table 6. Parameters of the experiments

Parameter	Value
The number of Co-Us in a Co-net	20
The number of task constraints for each Co-U	3~5
CE control factor ε	Assigned from 0.1 to 0.9
Total runs for each CE control factor ε	100 runs

5.4 Experimental Results

In the experiments, there are three values for the "type (form) of Co-net", and 9 for "CE control factor (ε)" ranging from 0.1 to 0.9. Therefore, there are 27 combinations for 2700 runs (each combination is run 100 times). According to the MANOVA analysis conducted (Table 7), the null hypotheses for each factor, CE control factor (ε) and type of Co-net, are rejected. Both ε and the type of Co-net are significant and affect the three output variables. In the following sections, we further discuss the experimental results based on the three output measures.

Table 7: 2 way--3 variables MANOVA analysis

Class	Levels	Values
Type (form) of Co-net	3	1, 2, 3
CE control factor (ε)	9	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
3 Variables: number of CE-infected Co-U ($\|\pi'\|$), conflict-severity (Δ), and communication time (Tc) Number of observations 2700		

The ANOVA Procedure
Multivariate Analysis of Variance
Characteristic Roots and Vectors of: E Inverse * H, where
H = ANOVA SSCP Matrix for Net
E = Error SSCP Matrix
Characteristic: Characteristic Vector V'EV=1

Root	Percent	# of CE-infected Co-U (π')	CE control factor (ε)	communication time (Tc)
1.52318443	85.58	-0.01245487	0.09268204	0.00479709
0.25675239	14.42	0.00673876	-0.06360035	-0.00050796
0.00000000	0.00	0.00359140	0.16293635	-0.00362291

MANOVA Test Criteria and F Approximations for the Hypothesis of No Overall **Co-net** Effect
H = ANOVA SSCP Matrix for Co-net
E = Error SSCP Matrix
S=2 M=0 N=1342.5

Statistic	Value	F Value	Num DF	Den DF	Pr > F
Wilks' Lambda	0.31535613	699.28	6	5374	<.0001
Pillai's Trace	0.80797374	607.32	6	5376	<.0001
Hotelling-Lawley Trace	1.77993682	796.97	6	3580.9	<.0001
Roy's Greatest Root	1.52318443	1364.77	3	2688	<.0001

NOTE: F Statistic for Roy's Greatest Root is an upper bound.
NOTE: F Statistic for Wilks' Lambda is exact.

Characteristic Roots and Vectors of: E Inverse * H, where
H = ANOVA SSCP Matrix for CFactor; E = Error SSCP Matrix; Characteristic: Characteristic Vector V'EV=1

Root	Percent	# of CE-infected Co-U (π')	CE control factor (ε)	communication time (Tc)
1.66186817	92.95	0.00095426	0.15347089	-0.00161157
0.12361014	6.91	-0.00680310	0.12269321	-0.00004582
0.00234388	0.13	-0.01289341	0.02400579	0.00581346

ANOVA Procedure
Multivariate Analysis of Variance
MANOVA Test Criteria and F Approximations for the Hypothesis of No Overall **CE control factor (ε)** Effect
H = Anova SSCP Matrix for CFactor; E = Error SSCP Matrix; S=3 M=2 N=1342.5

Statistic	Value	F Value	Num DF	Den DF	Pr > F
Wilks' Lambda	0.33356545	149.43	24	7793.7	<.0001
Pillai's Trace	0.73667398	109.40	24	8067	<.0001
Hotelling-Lawley Trace	1.78782219	200.08	24	5814.6	<.0001
Roy's Greatest Root	1.66186817	558.60	8	2689	<.0001

NOTE: F Statistic for Roy's Greatest Root is an upper bound.

Comparing the Average Communication Time (Tc) in the three Co-net forms

The MANOVA analysis indicates that both ε and the form of Co-net significantly affect the average communication time (Tc). Figure 15 shows the comparison of the average communication time (Tc) for the three forms of Co-nets. The results show that on average a higher ε leads to a rise of the Tc, and therefore, the higher ε makes the CE problems more likely to occur. Once a CE problem occurs, more Co-Us need to become involved and more communication time is needed for exchanging the CE announcements and requests for evaluation results.

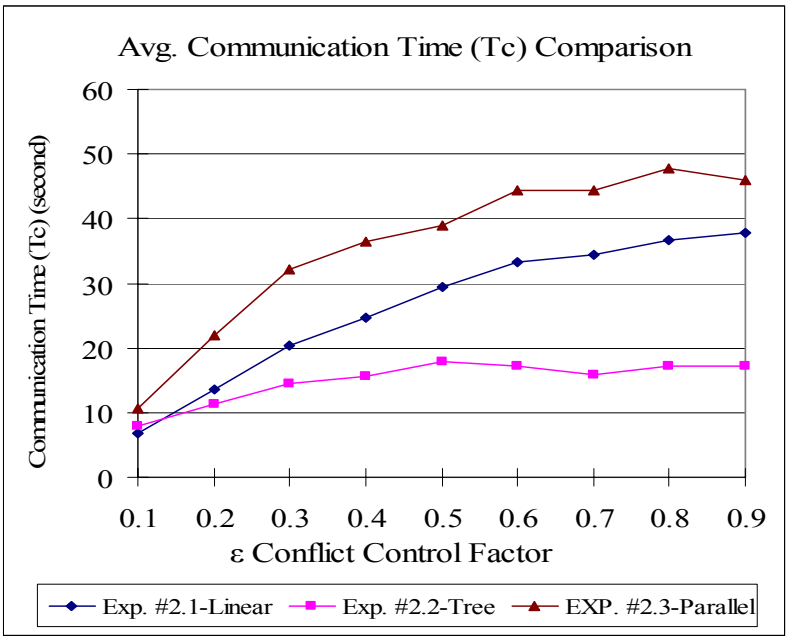


Figure 15. Average communication time (Tc) in three Co-net forms

For the same level of ε, two observations are important:

a. Parallel Co-net (Exp. #3) has on average the longest Tc because each Co-U must connect with more Co-Us in the parallel Co-net (diverging and merging).

Once a CE is detected by a Co-U, this Co-U has to exchange the current detection information with more than one Co-U both upstream and downstream. On the reverse, each Co-U in a linear Co-net needs to connect with only two Co-Us and each Co-U in a tree Co-net connects with only one Co-U, upstream.

b. Tree Co-net (Exp. #2) has the shortest *Tc* because the branch-structure of a tree Co-net leads to only one-way conflict propagation. More Co-Us will be located in the "leaves" of a tree Co-net, given the same total number of Co-Us. Hence, the Co-U in the "leaf" has a higher probability to be assigned to the initial CE problem. Most of the detection will be located in the regional area (branch of a tree) and the propagation is towards the root of the tree (Figure 16).

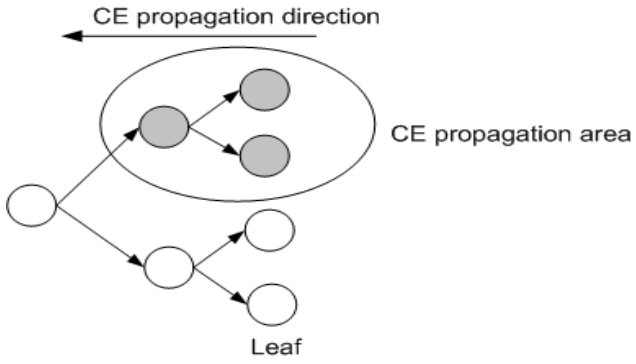


Figure 16. CE propagation area in a tree Co-net

Comparing the Average Number of CE-infected Co-Us *(|π'|)* in three Co-net forms

The MANOVA analysis indicates that both *ε* and the form of Co-net significantly affect the average number of CE-infected Co-Us (|*π'*|). A higher *ε* leads to a rise in the *(π')*. Furthermore, a higher *ε* increases the variance range of *Θ'*, and the CE will be more likely to occur. Figure 17 shows a comparison of the average of number of CE-infected Co-Us (|*π'*|).

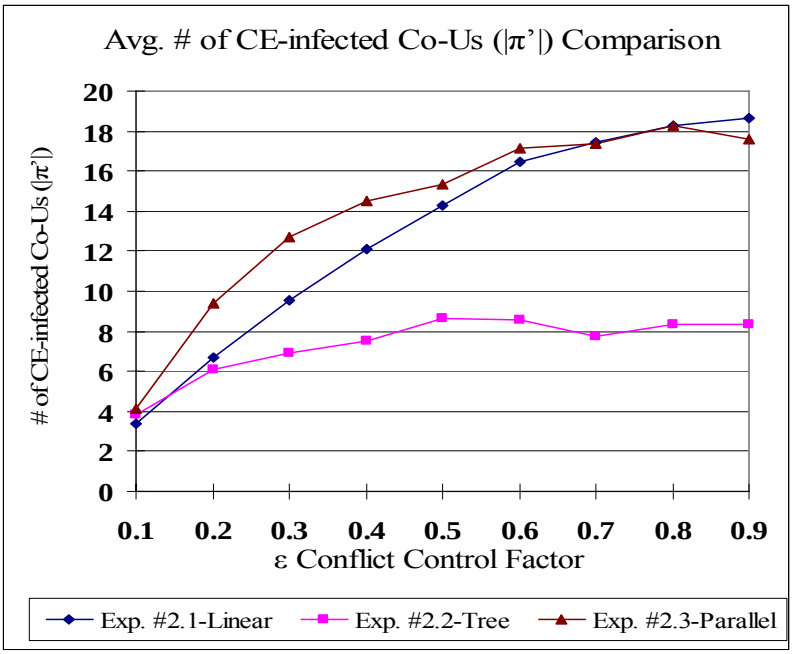


Figure 17. Average number of CE-infected Co-U (|π'|) in three Co-net forms

Based on the same level of ε, the results provide important attributes for the three Co-nets:

a. The linear and parallel Co-nets (Exp. #1 and #3) have on average a higher |π'|. More Co-Us in linear and parallel Co-nets are affected by conflict propagation than in tree Co-nets.

b. Tree (Exp. #2) has relatively less |π'|, since more Co-Us are located in the "leaves" of the Co-net. A tree Co-net has a greater probability of assigning an initial CE to a leaf-Co-U, and the conflict propagation is likely to be limited to only one direction. Also, the conflict propagation is likely to occur in a regional area (branch) in a tree Co-net. This observation is consistent since the tree Co-net has less Tc, since a smaller |π'| means less communication time is needed between Co-Us.

Comparing the Average Conflict-severity in the three Co-net forms

The MANOVA analysis shows that both ε and the type of Co-net significantly affect the average conflict-severity (Δ). The results show that a higher ε increases the average conflict-severity (Δ), since the higher ε makes the variance range of Θ' larger, and the CE problem will be more likely to take place. Once CE problems occur and more Co-Us get involved in this CE problem, more task constraints (Ω) cannot be satisfied by the Co-Us' changing current situations (Θ'), this makes the Δ (the summation of weights of unsatisfied task constraints) larger. Figure 18 shows the average conflict-severity (Δ) for the three experiments.

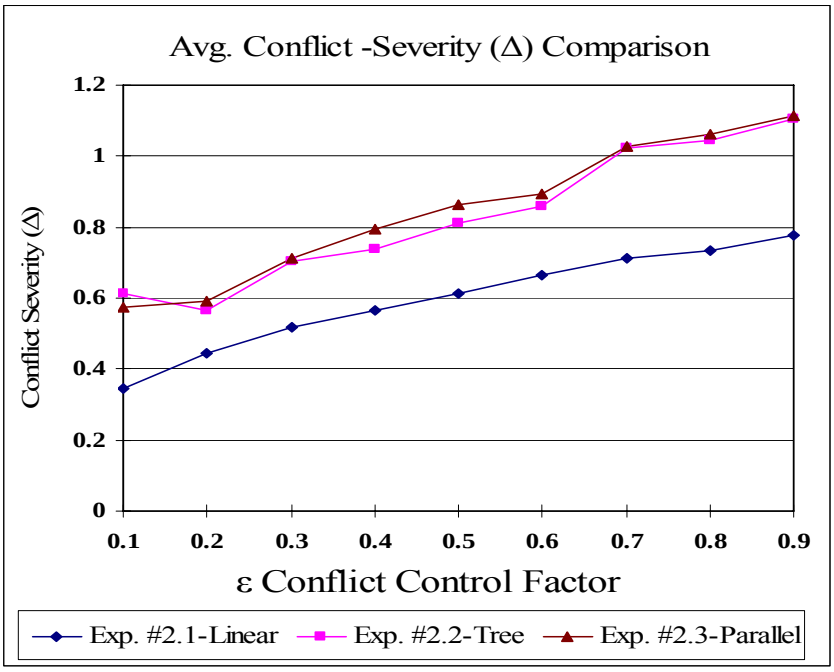


Figure 18. Average conflict-severity (Δ) in three Co-nets

Based on the same level of ε, the results convey important behaviors in these Co-nets:

a. Tree and parallel Co-nets (Exp. #2 and Exp. #3) have on average a higher Δ, as long as the ε increases. In a linear Co-net, each Co-U connects with only two neighboring Co-Us. In the more complex forms, each Co-U in a tree or parallel Co-net might have more than one neighboring Co-U in the downstream direction. Therefore, the average Δ in the tree or parallel Co-net is relatively higher and possibly higher than 1 when ε increases.

b. The Δ value in a linear Co-net (Exp. #1) is always less than or equal to 1 ($\Delta \leq 1$) since each Co-U is affected at most by only one other Co-U, therefore there is no add-up in each Co-U's Δ value.

c. Although $|\pi'|$ can represent how many Co-Us are affected by CE propagation, Δ measure is insufficient to show the seriousness of CE problems for each CE-infected Co-U.

6. Conclusions and Recommendations

In this research, a conflict/error detection protocol (CEDP) is developed for a distributed collaborative network. CEDP is expected to provide a detection-information exchanging mechanism for all collaborating members in a Co-net. Through the collaborative detection process, practically all possible CE problems that occur in a Co-net can be recognized. After analyzing and categorizing CE problems, we identify the CE table to serve as a basic data unit for storing CE information in a CE knowledge base. The CE knowledge base is a case-base repository that can be accessed by each Co-U in a Co-net to enhance the accuracy and quality of the detection process.

The answer for the research question "*Once a conflict or error occurs in a certain autonomous entity, how this event can affect other entities within the coordination network?"* is as follows. Conflicts and errors are unavoidable in a collaborative environment. In this research, we first review the definition of conflict and error, and distinguish them by illustrating the logical position within a Co-U boundary. Based on the definition, an important phenomenon, CE propagation, along with conflict and error are addressed in this report. CE propagation is a chain-effect of CE problems in a collaborative environment such as design teamwork or distributed manufacturing network. The occurred CE problem in a Co-U might propagate to other participants or partners and cause repeated cost for detecting and resolving it. Without sharing detection information among Co-Us, it is difficult to detect the CE propagation by an autonomous detector. Therefore, in this research, we not only focus on detecting CE problems that occur at an individual Co-U, but also on CE propagation problems in a Co-net.

Another research question addressed is: *How can a CE detection protocol (CEDP) be developed and specified?* In the proposed CEDP model, a Conflict/Error Detection Agent (CEDA) is designed as a distributed agent that is equipped to perform CE detections in a Co-U individually. Through the CEDP, CEDAs can communicate with each other to share CE information and detect CE propagation together. Besides, all CEDAs access the CE knowledge base and update the detection information. The cumulative detection information can be exchanged within a Co-net and provide useful "experience" and learning for handling similar CE detection.

The third research question posed is: *How can CE measurement be defined to represent the relative seriousness of a given CE event?* How to measure the detected CE problem is an important issue for CE detectors. In this research, a new conflict-severity (Δ) measure is defined and used to measure the degree of seriousness of a detected CE. It is defined as the summed weights (importance) of unsatisfied task constraints. If task constraints from other Co-Us cannot be satisfied by current states of a Co-U, a conflict occurs. The higher the conflict-severity is, the more inconsistencies between Co-Us exist. This measure is applied in the experiments to evaluate the seriousness of detected CE problems.

What is the purpose of the experiments and the value of the results? Because the performance of CE propagation detection process might be diverse in different forms of collaborative networks, in the experiments, we apply the CEDP model to evaluate and understand the detection operation in three basic forms of Co-nets. The conflict-severity measure is used to evaluate the detected CE problem. The experimental results show that in a relatively more complicated Co-net, such as parallel network, more Co-Us are involved in and affected by CE propagation, consequently more communication time is needed for detection. As a result, the conflict-severity (Δ) is also higher in more complicated Co-nets. It also means that the conflict-severity (Δ) can not only be used to measure the severity of each detected CE, but can also provide useful information to identify and analyze the degree of seriousness of CE propagation in a Co-net.

Recommended CEDP Design Guidelines

CEDP provides an information exchange mechanism to aid in detecting CE propagation. In terms of implementation, the following design guidelines are recommended:

a. Identify the functions of each detection agent (CEDA).

For each given CE problem, appropriate detection methods should be applied in the CEDA's detection policy. A DPEM mechanism can assist to select the best policy in given situations, based on relevant information gleaned across the distributed Co-net. Functions for sending and receiving messages are also needed for each CEDA to communicate with other CEDAs under the protocol.

b. Define the data structure for storing cumulative detection information.

   The description of the detected CE, occurrence time, CE infected member, and proper detection method are the main elements to be included and stored to support the detection process.

c. Apply the conflict-severity measure to assess the degree of seriousness of a detected CE problem and its potential damaging impact on other C-Us via propagation.

d. Define the message routing mechanism to transmit exchanged information between Co-Us. An efficient and reliable exchange across the Co-net will be necessary to assure timely processing and response during the detection process.

Future Research

Conflict/error detection is an initial stage of conflict/error management [Balakrishnan et al., 1994]. Solving or recovering the detected CE problem in the collaboration environment is a following mission. Several undeveloped areas which need to be investigated are:

a. A well-organized conflict/error knowledge base is necessary to improve the conflict/error detection and resolution process. Knowledge acquiring and query mechanisms that enable collaborative members to deposit and retrieve detection information are important issues.

b. A sophisticated distributed conflict/error management that contains the conflict/error resolution protocol is needed for autonomous members to solve detected CE problems under coordination. An effective communication mechanism to improve the quality of CE resolution is a crucial issue.

c. Evaluation and comparison of the proposed CEDP with other detection protocols will be the next step to appraise performance of CEDP.

References

Albright A. C., 2001, VBA for Modelers – Developing Decision Support Systems with Microsoft Excel, Duxbury/Thomson Learning, Pacific Grove, CA, U.S.A.

Anussornnitisarn, P., 2003, Design of Active Middleware Protocols for Coordination of Distributed Resources, Dissertation for Ph.D., Purdue University, West Lafayette, IN., U.S.A.

Anussornnitisarn, P., and Nof, S. Y., 2001, The Design of Active Middleware for e-Work Interactions, *Research Memorandum*, School of Industrial Engineering, Purdue University, West Lafayette, IN., U.S.A.

Anussornnitisarn,P., and Nof, S.Y., 2003, e-Work: The Challenge of the Next Generation ERP Systems, *Production Planning and Control*, v. 14, n 8, pp. 753-765.

Anussornnitisarn, P., Nof, S. Y. and Etzion, O., 2001, Analysis of Cooperative Tasks for Networked Enterprise's Middleware, *Research Memorandum*, School of Industrial Engineering, Purdue University, West Lafayette, IN., U.S.A.

Bakken, D. E., 2003, Middleware, Chapter in *Encyclopedia of Distributed Computing, Urban J. and Dasgupta P.(editors)*, Kluwer Academic Publishers.

Bishop, T. A., and Karne, R. K., 2003, A Survey of Middleware, *18th International Conference on Computers and Their Applications*, March 26-28, 2003, Honolulu, Hawaii, U.S.A.

Hagen, C., and Alonso, G., 2000, Exception Handling in Workflow Management Systems, *IEEE Transaction on Software Engineering*, v 26, n 10, pp. 943-958.

Khanna, N., Fortes J. A. B., and Nof, S.Y., 1998, A Formalism to Structure and Parallelize the Integration of Cooperative Engineering Design Tasks. *IEE Transactions*, v 30, n 1, pp. 1-15.

Klein, M., 2000, Towards a Systematic Repository of Knowledge About Managing Collaborative Design Conflicts, *Proceedings of the Sixth International Conference on Artificial Intelligence in Design*, Worcester, MA, June 26-29, 2000.

Klein, M., and Dellarocas, C., 2000, A Knowledge-based Approach to Handling Exceptions in Workflow Systems, *Computer Supported Cooperative Work*, v 9, pp. 399-412, Kluwer Academic Publishers.

Lara, M. A., and Nof, S. Y., 2003, Computer-supported Conflict Resolution for Collaborative Facility Designers, *International Journal of Production Research*, v 41, n 2, pp. 207-233.

Li, X., Zhou, X., and Ruan, X., 2002, Conflict Management in Closely Coupled Collaborative Design System, *Int. J. Computer Integrated Manufacturing*, v 15, n 4, pp. 345-352.

Luo, Z., Sheth, A., Kochut, K., and Arpinar, B., 2000 ,Exception Handling in Workflow Systems, *Applied Intelligence*, v 13, n 2, pp. 125-147, Sept.-Oct. 2000.

Luo, Z., Sheth, A., Kochut, K., and Arpinar B., 2003 ,Exception Handling for Conflict Resolution in Cross-Organizational Workflows, *Distributed and Parallel databases*, v 13, n 2, pp. 271-206, May 2003.

Malone, T. W., Crowston, K., Lee, J., Pentland, B., Dellarocas C., Wyner G., Quimby J., Osborn C. S., Bernstein A., Herman G., Klein M., and O'Donnell E., 1999, Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. *Management Science*, v 45, n 3, pp. 425-443.

Tan, G. W., Hayes, C. C., and Shaw, M., 1996, An Intelligent-Agent Framework for Concurrent Product Design and Planning, *IEEE Transactions on Engineering Management*, v 43, n 3, pp. 297-306.

Yang, C-L., Conflict and Error Detection Protocol with Active Middleware, M.S.I.E. Thesis, Purdue University, August 2004.